difficult, due partly to the fact that instances of the class implementations are created only at run-time. Class::MethodMaker, goes some way to help this by providing a way to obtain a reference to the parent object of a given object. By itself however, this is of limited value.

[**0054**] In one embodiment of the present invention a mechanism is provided which enables a hierarchical tree of objects to be navigated through by making use of functionality provided by Class::MethodMaker, as described below.

[**0055**] This navigation functionality may be provided, for example, through a tree navigation class which provides a method which accepts as an input string the relative location of an attribute in a hierarchical arrangement (or tree) of objects. For example, referring back to **FIG. 1**, suppose that the refresh rate attribute **112** of the monitor object **108** is dependent of the pixel clock attribute **107** of the video card object **104**. From within the monitor class declaration, a string is defined giving the relative position within the tree of the pixel clock attribute **107**. For convenience, such a string may use notation similar to that well known from Unix and MS-DOS file systems. For example the string:

[**0056**] ..\..\video_memory

[**0057**] may be used to indicate that the parameter video memory is found in the object 'two levels up' from the current object. The first level up being the monitor object **108** and the second level up being the video card object **104**

[**0058**] Similarly, the string:

[**0059**] ..\..\..\processor\cache_size may be used to indicate that the parameter cache_size is an attribute of the processor object which is located 'three levels up' from the current monitor object and 'in the processor object'.

[**0060**] An embodiment of this tree navigation functionality is shown in flow diagram form in **FIG. 2**. The first step **200**, is to obtain the string defining the location of the required attribute and to parse the string to extract the path information and the attribute required. Hereinafter, the required attribute is referred to as a warp master value. Using the tree navigation function the tree of objects may be 'navigated', steps **202** and **204**, until the required object is reached. When the destination object is reached, the required warp master value is obtained, step **206**, and the warp master value is returned to the requesting object, step **208**.

[**0061**] Additional dependency functionality between classes may be built on top of the tree navigation function, as will be described further below.

[**0062**] Embodiments of the present invention provide for a declarative manner for defining three main types of dependency relationship, as will be described below. The first dependency type is where an attribute of one class (the warp value) is dependent on the value of an attribute in another class (the warp master value). The nature of this dependency is such that should the warp master value change, the warp value is automatically updated.

[**0063**] An embodiment outlining the mechanisms by which the warp value dependency may be implemented will now be described below.

[**0064**] **FIG. 3** is a diagram showing an extract of the configuration tree shown in **FIG. 1**. The dotted line **302** indicates that the attribute refresh rate, **112**, (the warp value) of the monitor object **108** is dependent on the attribute pixel clock, **107**, (the warp master value) of the video card object **104**. In order to define the nature of the dependency it is necessary to define the relative location of the warp master value, and the rule (hereinafter referred to as the warp rule) to be applied to the warp value. For this example, assume that the relationship between the refresh rate and the pixel clock attribute is defined as: if the pixel clock value is greater than 100 Mhz, then the maximum r fresh rate is 50 Hz, if the pixel clock value is greater than 200 Mhz then the maximum refresh rate is 60 Hz, and if the pixel clock value is greater than 300 Mhz then the maximum refresh rate is 70 Hz.

[**0065**] As previously described, the warp master value may be defined as:

[**0066**] ..\..\pixel_clock

[**0067**] The warp rule may, for example, be expressed as:

[**0068**] If warp_master_value >'100', max ='50';

[**0069**] If warp_master_value >'200', max ='60';

[**0070**] If warp_master_value >'300', max ='70';

[**0071**] A warp value attribute therefore has additional associated information which defines the location (the warp master value) and the nature of (the warp rule) the dependency. One way in which this dependency information can be associated with an attribute is to declare the attribute as of an object type having appropriate data containers and accessor methods for implementing the required functionality. For example, below is shown an example Value class which may be used for this purpose:

VALUE CLASS - TABULATED VIEW
CLASS NAME: Value

ATTRIBUTES	Type	Comments
name	created depending on type specified	
min	Integer	
max	Integer	
warp__master__value	String	
warp__rule	String	

[**0072**] Thus, an attribute of type Value has data containers for holding, amongst others, the name and type (e.g. integer, enum etc.), any maximum or minimum values, and any associated warp information. The accessor methods provided by the Value class may use this information, for example, for ensuring that an attribute value is within the limits defined by the maximum and minimum values. Similarly, the Value class accessor methods also provide the required functionality to interpret a warp rule from a string and to transform this into the correct processing steps as defined by the warp rule. In this way, much of the complex functionality may be hidden from the user, for example, by the Value object. For clarity, the following examples do not show each attribute as being of a type Value, although it will be appreciated that any attribute having associated warp information may advantageously be declared as so.

[**0073**] It should be noted that the Value class itself may not necessarily be created by a class-making module, and is preferably defined using conventional techniques.